Das Frontend aus der Datenbank

Dominic Weiser
ISTEC Industrielle Software-Technik GmbH
Nobelstraße 12, 76275 Ettlingen

Schlüsselworte
Java EE, JSF, Primefaces, Eclipselink, Atmosphere, CDI, Oracle DB

Einleitung
GUI-Frameworks gibt es wie Sand am Meer. Jedes ist schneller, besser und einfacher zu handhaben als das andere. Zumindest lauten oft so die Marketingaussagen der Hersteller. Kurze „Getting Started"-Beispiele sollen dies belegen. In der Regel klappt das auch sehr gut. Schnell werden erste Ziele erreicht. Jedoch: Was passiert, wenn mit diesem Framework eine Anwendung erstellt werden soll? Erst dann werden die Stärken und besonders auch die Schwächen des Frameworks sichtbar.
Als IT-Dienstleister sind wir direkt von den Stärken und Schwächen der GUI-Frameworks betroffen. Einfachste Darstellungen von Daten erfordern Anpassungen und somit Programmieraufwand. Das hat wiederum ein Update der produktiven Applikationen zur Folge. Dies ist besonders dann kritisch, wenn jede Downtime der wertschöpfenden Prozesse Geld kostet.
Um diese Kernprobleme zu lösen, hat die ISTEC einen GUI-Baukasten auf Primefaces-, Java- und Oracle DB-Basis erstellt, mit welchem sich Dialogänderungen mehrheitlich in der Datenbank konfigurieren lassen.

Architektur
Damit die Informationen der Datenbank zur Oberfläche (Browser) durchgereicht werden können, bedienen wir uns diverser Zugriffsschichten. Durch diese Trennung der Themengebiete wird es ermöglicht auch später einfach einzelne Bereiche auszutauschen und somit neue Technologien einzusetzen.

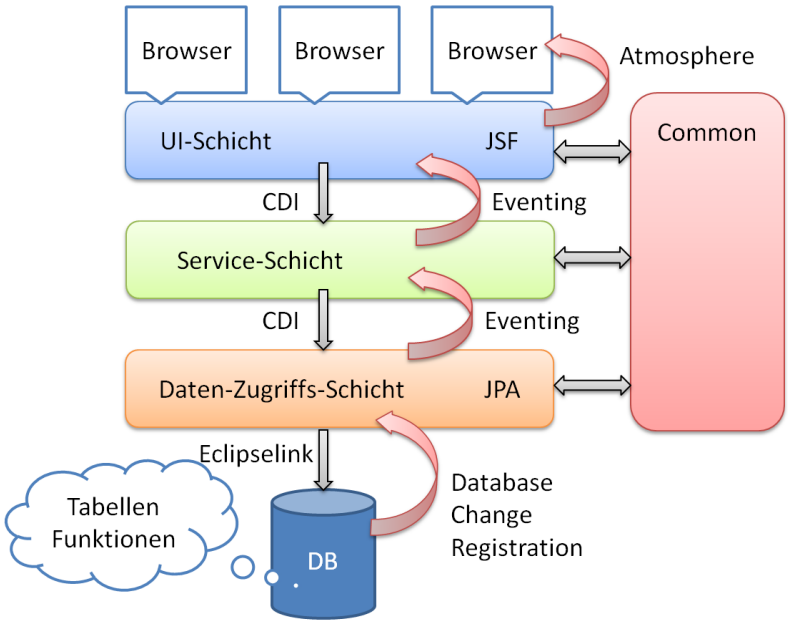

Abb. 1: Architektur

Vor Beginn der Toolentwicklung wurde JSF / Primefaces auf Grund der vorhandenen benötigten Komponenten als neues UI-Framework ausgewählt. Auf die Verwendung der UI-Komponenten wird im nächsten Kapitel eingegangen.
Wie voran gegangen beschrieben setzt sich das Framework aus mehreren Schichten zusammen. Diese sind mittels CDI miteinander verbunden. Somit ist gewährleistet, dass die jeweiligen Implementierungen mittels Annotations austauschbar sind. Der Zugriff auf die Datenbank wird mittels JPA/Eclipselink realisiert. Hier kommen drei verschiedene Konstrukte zum Einsatz:

1. JPA verwaltetet Entitäten
   Diese werden für alle Tabellen verwendet, welche das Aussehen der Oberflächen steuert (siehe UI-Konfiguration)
2. JPA Session Pooling
   Mittels JPA wird der DB-Session-Pool verwaltet. Es werden direkt Datenbank-Funktionen aufgerufen. Dieses Konstrukt kommt immer dann zum Einsatz, wenn Nutzdaten gelesen und geschrieben werden. (Siehe *LoadMethod* UI-Komponenten)
3. Direktverbindung
   Dieses Konstrukt kommt für das Eventhandling zum Einsatz. Auf einer Tabelle werden alle Events gespeichert, welche im Hintergrund geworfen werden. Mittels Change Registration werden diese Änderungen direkt an den Server übermittelt.

Durch diese Trennung ist es uns möglich nahezu alle Anpassungen der Oberfläche und der damit verbundenen Logik, welche fast ausschließlich auf der Datenbank zu finden ist, ohne Deployment und somit im laufenden Betrieb durchzuführen. Nur vereinzelte Spezialdialoge benötigen ein gesondertes Deployment.

UI-Komponenten
Nun reichen die Trennung in Schichten und der Einsatz moderner Frameworks nicht aus, um dem Anwender die Anwendung am Frontend darzustellen. Hier kommen JSF, Primefaces und Java EE ins Spiel. Gerade Primefaces bietet hier eine Vielzahl reicher Komponenten. Mit Hilfe des Composite-Components (zusammengesetzte Komponenten) Konzepts lassen sich diese Komponenten sehr leicht an die eigenen Bedürfnisse anpassen. In unserem Fall muss die Komponente wissen, welche Datenbankfunktion für das Laden (*LoadMethod*) und Speichern (*ExecuteMethod*) der Daten zuständig ist. Die Komponenten wurden so erweitert, dass sie sich als abhängige Komponente an eine andere registrieren können. Ändert sich der Inhalt einer Komponente, werden auch alle abhängigen aktualisiert.

```
<cc:interface>
  <cc:attribute name="loadMethod" required="false"/>
  <cc:attribute name="loadParameter" required="false"/>
  <cc:attribute name="requiredComponents" required="false/>
  <cc:attribute name="eventname" required="false"/>
  <cc:attribute name="metaMethod" required="false"/>
</cc:interface>

<cc:implementation>
  <p:importConstants type="de.istec.pls.client.ui.ComponentConstant" var="ComponentConstant" />
  <p:inputText id="inputText" widgetVar="#{cc.clientId}" onkeypress="enterAsTab(this,event);"
                value="#{cc.attrs.compBean.componentData[cc.clientId].value}"
                required="#{cc.attrs.compBean.componentData[cc.clientId].isRequired()}" >
    <f:event listener="#{cc.attrs.compBean.initializeAndLoad}" type="preRenderComponent" />
    <f:attribute name="#{ComponentConstant.COMPONENT_ID}" value="#{cc.clientId}" />
    <f:attribute name="#{ComponentConstant.LOAD_METHOD}" value="#{cc.attrs.loadMethod}" />
    <f:attribute name="#{ComponentConstant.LOAD_PARAMETER}"
```

```
                value="#{cc.attrs.loadParameter}" />
   <f:attribute name="#{ComponentConstant.COMPONENT_TYPE}" value="inputText" />
   <f:attribute name="#{ComponentConstant.EVENTNAME}" value="#{cc.attrs.eventname}" />
   <f:attribute name="#{ComponentConstant.METAMETHOD}" value="#{cc.attrs.metaMethod}" />
  </p:inputText>
<cc:implementation>
```

Das Snippet zeigt Auszüge aus der Composite-Component Inputtext. Im Interface wurden die voran beschriebenen Attribute wie *LoadMethod, Eventname* oder abhängige Komponenten (*requiredComponents*) definiert. Diese werden mittels Expression-Language an die Komponenten Implementierung bzw. die Komponentenbean weitergereicht. Dies hat zur Folge, dass der Einsatz der InputText-Komponente am Frontend sehr schlank möglich ist.

```
<pls:inputText id="PDAVERSION"  loadMethod="uiPDAVersion" eventname=“IDChange“
                requiredComponents="pdapunkt:PDAID" loadParameter="pdapunkt:PDAID"/>
```

Dieser Inputtext ist nun so konfiguriert, dass er seine Daten aus der Datenbank-Funktion *uiPDAVersion* bezieht. Diese Methode benötigt die Inhalte der Komponente *PDAID*. Die Inhalte der abhängigen Komponenten werden als JSON-Objekt zusammen mit KomponentenID und Inhalt im JSON-Array übermittelt.

Zusätzlich wurde die Komponente auf den Eventname *IDChange* registriert. Auf dem Server liegt ein Listener-Bean, welche die Komponenten abhängig zu ihrem Eventname registriert. Diese Bean wird mittels Database-Change-Registration informiert, sobald ein neuer Wert auf der Event-Tabelle eingetragen wird. Dies ist der Auslöser für die Events auf Datenbank-Seite. Wurde ein Event für eine Komponente gefeuert, aktualisier sich diese und bezieht ihre Daten über die *LoadMethod* neu. Zusätzlich kennt die Komponente alle von ihr abhängigen Komponenten und veranlasst, dass auch diese sich aktualisieren.

UI-Konfiguration

Nachdem im vergangenen Kapitel auf den Umbau der Primefaces Komponenten eingegangen wurde, widmen wir uns nun dem Zusammensetzen der einzelnen XHTML-Seiten zu einer Anwendung. Grundsätzlich besteht eine Anwendung aus vielen einzelnen Seiten (*AppDialog*). Jede Seite muss über das Menü (*AppMenu*), eine Schaltfläche auf einem Dialog oder direkt mit dem Link erreichbar sein. Wobei der Menüeintrag bzw. die Schaltfläche nur die grafische Aufbereitung des Links sind. Ein Link setzt sich immer aus der Domäne, dem Anwendungspfad, der XHTML-Seite und der *DialogId* zusammen. Mit Hilfe der Seite wird das benötigte Template vom Server geladen und anhand der *DialogId* (Standard-Dialog verweis auf Datenbanktabelle) bzw. Lade-Methode mit Daten befüllt.
Was bedeutet dies für die Konfiguration? Eine Seite muss auf der Datenbank mit der *DialogId*, dem XHTML-Template und ggf. der Datenquelle konfiguriert werden. Das XHTML-Template wird benötigt, damit das Menü den Pfad zusammensetzen kann.
Nachdem der Anwender mit Hilfe des Links auf die gewünschte Seite gelangt ist, müssen seine Anwendungsmöglichkeiten über das Berechtigungskonzept (*AppRight*) abgesichert werden. Hier stehen dem Entwickler verschiedene Aktionen (Lesen, Schreiben, Löschen) zur Verfügung. Im Falle einer Tabelle wird die Berechtigung bis auf Spaltenebene heruntergebrochen. Hier wird die Berechtigung für verschiedene Detailgrade (Keine Anzeige, Liste oder Detailansicht) spezifiziert. Zusätzlich kommt hier der Anlege- und Duplizier-Modus in den Einstellungen hinzu.

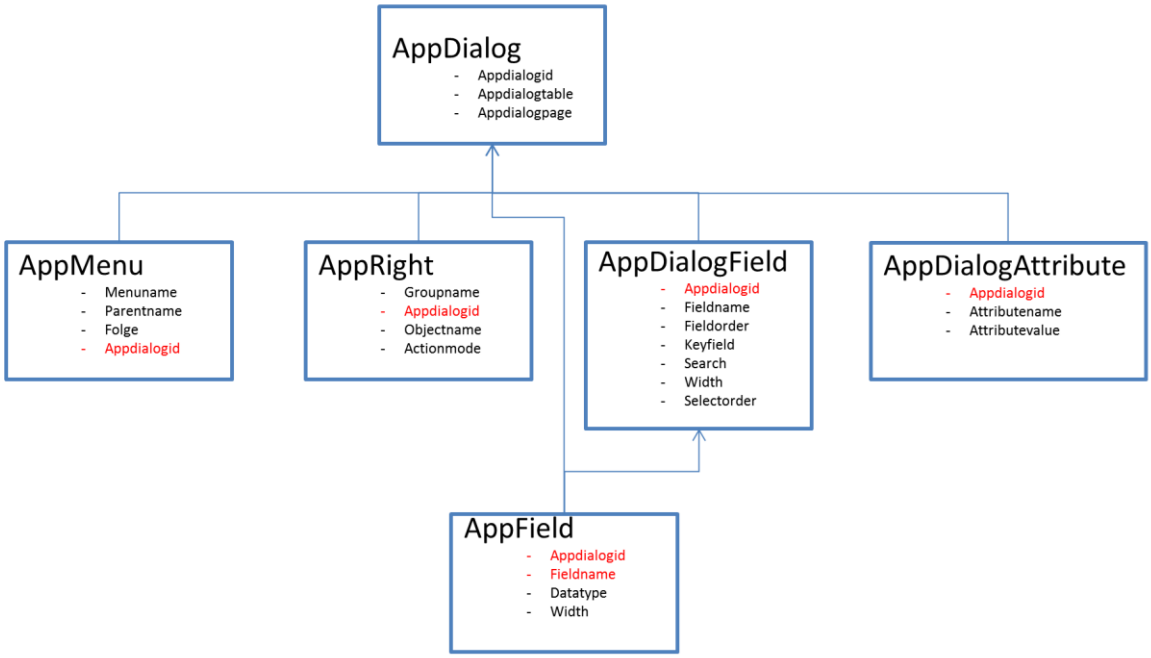

Abb. 2: AppDialog

Zum Zeitpunkt der Entwicklung des Frameworks hat sich herausgestellt, dass die Masse der benötigten Dialoge reine Abbildungen von Datenbank-Tabellen oder Views sind. Daher ist das Herzstück jeder Anwendung der so genannte Standard-Table-Dialog. Um diesen zu konfigurieren werden zusätzliche Einstellungen (*AppDialogField*, *AppField* und *AppDialogAttribute*) benötigt. Im *AppDialogField* werden die einzelnen Tabellenspalten definiert. Hier kann der Spalte neben einer eigenen Sortierreihenfolge auch die Information des Schlüssels sowie die Art der Suche eingestellt werden. Weitere Einstellungen lassen sich mit Hilfe des *AppFields* vornehmen. Die wichtigsten Einstellungen sind die des Datentyps und des damit verbundene Datapattern. Eine individuelle Lookupmethode kann aus Abkürzungen leserliche Texte machen (z.B. DB-Eintrag ‘N‘ zu ‘Nein‘ übersetzen). Neben der eindeutigen Definition mit Hilfe der *AppDialogId* und dem *Fieldname* können auch Feldern über die Grenzen des *AppDialogs* hinweg konfiguriert werden (Wildcard ‘_‘).
Das Erscheinungsbild der Tabelle lässt sich in der *AppDialogAttribute* einstellen. Hier kann zum Beispiel definiert werden, wie viele Zeilen pro Seite angezeigt werden, ob es einen Paginator oder ein Datenexport gibt.

Kontaktadresse:
Dominic Weiser
ISTEC Industrielle Software-Technik GmbH
Nobelstraße 12
D-76275 Ettlingen

Telefon:	+49 (0) 7243-7005 164
Fax:	+49 (0) 7243-7005 199
E-Mail	Dominic.Weiser@istec.de
Internet:	www.istec.de